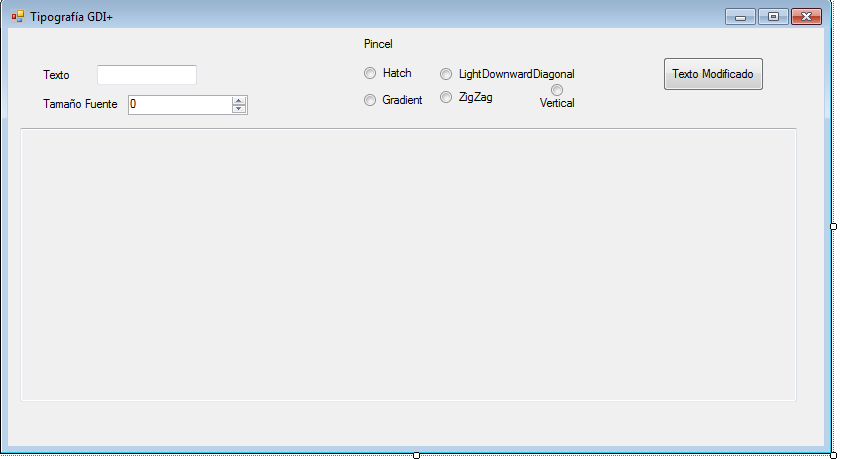
GD+



**Introducción**

Antes de continuar hemos de advertir que las líneas de código que exponemos en este tutorial son opciones entre toda la sobrecarga que ofrece VSNET. Abarcarla toda sería demasiado extenso; de todas formas no es difícil una vez entendidos los conceptos básicos.

Las clases que producen gráficos vectoriales en VSNET están contenidas en los espacios System.Drawing y System.Drawing.Drawing2D:

Imports System.Drawing

Imports System.Drawing.Drawing2D

**El lienzo**

El primer paso para poder dibujar gráficos es crear u obtener un objeto Graphics, que viene a ser el lienzo donde expresaremos nuestro arte. El objeto Graphics carece de constructor público, por tanto no se puede instanciar con la palabra clave New.

El evento Paint contiene una referencia a Graphics en su parámetro e:

Private Sub Form1\_Paint(ByVal sender As Object, ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

Dim gr As Graphics = e.Graphics

End Sub

También puede obtenerse llamando al método CreateGraphics. El siguiente ejemplo crea tres objetos Graphics en el formulario, en un PictureBox y en un botón:

Dim Lienzo1 As Graphics = Me.CreateGraphics

Dim Lienzo2 As Graphics = PictureBox1.CreateGraphics

Dim Lienzo3 As Graphics = Button1.CreateGraphics

La diferencia entre obtener Graphics del objeto e y del formulario o control radica en el momento en que se generará el dibujo. El evento Paint se dispara siempre que sea necesario repintar cualquier área del formulario o control. Si nuestro dibujo es demasiado complejo, pueden producirse destellos desagradables en la pantalla. En cambio, obtener Graphics del control o formulario otorga al programador un control absoluto del momento en que se pinta el dibujo.

Borrar un dibujo no es tan fácil como crearlo. El código utiliza objetos para trazar líneas y colorear formas sobre, por ejemplo, un formulario, pero las líneas y las formas pintadas no son objetos, sino puntos coloreados, por tanto no se pueden modificar, a no ser que los borremos y volvamos a dibujar. Simulamos, por tanto, el borrado repintando puntos con el color del lienzo, es decir, dibujando o pintando lo que queremos borrar con el color de fondo del formulario o del control del que hemos obtenido el objeto Graphics.

Graphics sólo nos ofrece un método de borrado: .Clear(Color), que aparentemente borra la superficie completa pero en realidad la cubre con el color especificado:

gr.Clear(Me.BackColor)

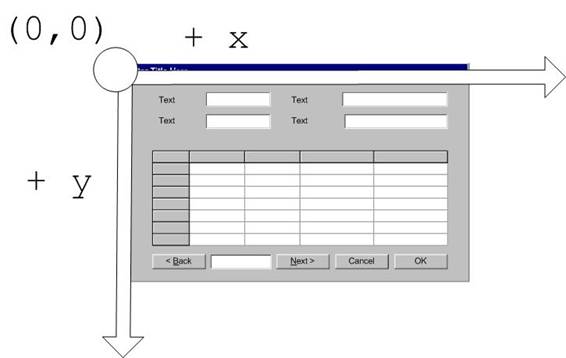
Una vez utilizado el objeto Graphics es conveniente cerrarlo invocando el método Dispose

gr.Dispose()

**Los ejes de coordenadas**

Cualquier formulario o control contiene tres sistemas de coordenadas, llamados “coordenadas de dispositivo”, “coordenadas de página” y “coordenadas de mundo”.

En los tres sistemas los valores +X aumentan hacia la derecha y los valores +Y aumentan hacia abajo, según muestra el siguiente esquema:



Las coordenadas de dispositivo y las coordenadas de página tienen su origen inamovible en el extremo superior izquierdo del área cliente del formulario o control. El origen de as coordenadas de mundo, en cambio, aunque inicialmente se sitúa en el extremo superior izquierdo del área cliente, puede moverse a cualquier punto del área cliente del formulario o control. En el capítulo dedicado a las transformaciones aprenderemos a hacerlo. Aquí nos limitaremos a distinguir los tres sistemas de coordenadas que conviven en todo formulario o control.

Coordenadas de dispositivo (device coordinate system) Son las coordenadas por defecto inamovibles y medidas en píxeles

Coordenadas de página (page coordinate system). También son inamovibles, pero admiten diferentes unidades de medida, que se establecen llamando al método PageUnit del objeto Graphics. Este método toma como argumento un miembro de la enumeración GraphicsUnit, que especifica la unidad de medida que se utilizará:

|  |  |
| --- | --- |
| **Display** | 1/75 pulgadas. |
| **Document** | 1/300 pulgadas. |
| **Inch** | Pulgada. |
| **Millimeter** | Milímetro. |
| **Pixel** | Pixel. |
| **Point** | Punto de impresión (1/72 pulgadas) |
| **World** | La misma que se haya definido para las coordenadas de mundo. |

El valor por defecto es Pixel, de modo que, si no se modifica, las coordenadas de página coinciden con las de dispositivo.

Coordenadas de mundo (world coordinate system). Salvo en la orientación de los valores X e Y, que en ningún caso se puede cambiar, son totalmente personalizables: pueden moverse y girarse y se pueden definir unidades de medida personalizadas. Las coordenadas de mundo son las utilizadas por el objeto Graphics. Si no se personalizan, que veremos en el capítulo Transformaciones cómo hacerlo, coinciden con las coordenadas de página.

**El color**

No vamos a entrar en teoría del color, sino sólo indicaremos cómo se definen los colores en VBNET

Todo color se compone de tres ingredientes: rojo, verde y azul, y un nivel opcional de transparencia. Cada ingrediente toma un valor Integer en el rango 0-255. 0 indica ausencia de color y 255 total presencia. (0, 0, 0) es el color negro y (255, 255, 255) el color blanco. La transparencia también se mide de 0 a 255: 0 indica transparencia total y 255 opacidad total.

Además, cada color tiene asociado un número Integer con signo negativo.

VSNET nos ofrece la estructura Color que nos permite utilizar colores predefinidos con sólo escribir su nombre: Color.Blue, Color.Green, etc. Color es una estructura, no una clase, por tanto no admite instanciación con New.

A continuación expondré esquemáticamente los métodos de la estructura Color utilizados en este tutorial:

Color.FromArgb Define un color tomando como parámetros los valores para rojo, verde y azul y, opcionalmente, el nivel de transparencia. También puede utilizarse con sólo el número asociado al color. Ejemplo

Color.FromArgb(77, 77, 77)

Color.FromArgb(100, 77, 77, 77) ‘Con nivel de transparencia

Color.FromArgb(-123456) ‘Número del color

Color.ToArgb Devuleve el número asociado al color:

Dim n As Integer = Color.Blue.ToArgb()

A, R, G, B Devuelve, respectivamente, los niveles de transparencia, rojo, verde y azul

Dim rojo As Integer = Color.Blue.R()

FromName Define un color a partir de su nombre

Dim MiColor As Color = Color.FromName(“Blue”)

**El lapicero**

Antes de dibujar tenemos que definir un lapicero o “pen”. Al construirlo podemos definir sólo el color o el color y el grosor:

Dim Lápiz As New Pen(Color.Blue)

Dim Lápiz As New Pen(Color.Blue, 10)

También podemos crear un lapicero sin construirlo llamando a la enumeración Pens.

**Formas Básicas**

**Punto**

Un punto es un objeto compuesto por un par de valores de tipo Integer que representan las coordenadas X e Y en el eje de coordenadas. Se construye así (para un punto situado en x=10 y y=10:

Dim MiPunto As Point = New Point(10, 10)

No hemos encontrado ningún método que dibuje un punto en una superficie en blanco. Un buen truco es dibujar una línea cuyos extremos inicial y final sean tan cercanos que parezca un punto, o también pintar un círculo muy muy pequeñito.

Otra cosa es cambiar u obtener el color de un pixel de un bitmap. [Aquí](http://www.elguille.info/colabora/puntoNET/NieveEnVB.htm) tenéis un buen ejemplo de ello.

Las formas primitivas que explico a continuación se dibujan invocando métodos de la clase Graphics.

**Línea**

Una línea se define por dos puntos y el trazo que los une. El método que dibuja líneas se llama DrawLine:

Dim Lápiz As New Pen(Color.Blue, 10)

Dim Punto1 As Point = New Point(10, 10)

Dim Punto2 As Point = New Point(100, 100)

Lienzo.DrawLine(p, t1, t2)

Utilizando la notación sin variables intermedias quedaría así:

Lienzo.DrawLine(New Pen(Color.Blue, 10), New Point(10, 10), New Point(100, 100))

Junto con el método DrawLine disponemos del método DrawLines. En vez de dos puntos, DrawLines recibe un array de puntos y dibuja una línea quebrada compuesta por todas las líneas rectas que unen los puntos del array. Si queremos, sin embargo, que la unión entre todos los puntos forme una línea curva, disponemos del método DrawBeziers, que también toma como parámetro un array de puntos. Este método es ideal para trazar gráficas de funciones. El siguiente ejemplo traza un tramo de la gráfica de la función y=2x2+2x+2

Dim i As Short

Dim Puntos(100) As Point

For i = 0 To 100

Puntos.SetValue(New Point(i + 100, 2 \* i ^ 2 + 2 \* i + 2), i)

Next

Lienzo.DrawBeziers(Pens.Black, Puntos)

Observe el lector que el lapicero no ha sido construido, sino que se ha recurrido a la enumeración Pens.

Ahora que sabemos dibujar líneas podemos profundizar en el objeto lapicero.

Una línea puede ser continua (valor por defecto) o discontinua. Los diferentes modelos de discontinuidad se seleccionan llamando a la enumeración DashStyle:

Lápiz.DashStyle = DashStyle.DashDotDot

Podemos crear modelos de discontinuidad personalizados asignando a la propiedad DashPattern del objeto Pen un array de valores de tipo Single. El número de elementos de este array ha de ser par. Veamos un ejemplo:

Dim Modelo() As Single = {2, 8, 1, 9, 3, 8}

Lápiz.DashPattern = Modelo

'Dibuja un círculo

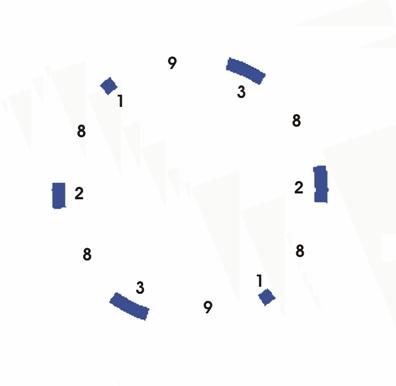
Lienzo.DrawEllipse(Lápiz, New Rectangle(200, 200, 200, 200))

‘Más abajo explico el método DrawEllipse

Este código genera el siguiente dibujo:

****

Cadapar de elementos del array representa una proporción entre un tramo pintado y el contiguo en blanco. En el dibujo siguiente he señalado las proporciones:



Además de trazar líneas discontínuas, el objeto Pen puede modificar los extremos de las líneas para convertirlas en flechas. La propiedad StartCap personaliza el origen de la flecha y la propiedad EndCap el final. Ambas esperan un valor de la enumeración LineCap. El código siguiente convierte una recta en la flecha siguiente:

Dim Lápiz As New Pen(Color.DarkOrange, 10)

Lápiz.StartCap = LineCap.SquareAnchor

Lápiz.EndCap = LineCap.ArrowAnchor

Lienzo.DrawLine(p2, 280, 30, 500, 30)

http://www.elguille.info/colabora/puntoNET/tutGDI/imagenes/image008.jpg

**Rectángulo**

Para VSNET, un rectángulo es un polígono de cuatro lados tal que, dado un lado cualquiera, existe otro paralelo a él, y además todos los vértices forman un ángulo recto. Un cuadrado, por tanto, sólo es un caso particular de rectángulo que se distingue porque todos sus lados miden lo mismo.

El objeto Rectangle se compone de las coordenadas de su vértice superior izquierdo, la anchura y la altura. Para dibujarlo invocamos al método DrawRectangle que acepta como parámetros el lapicero y un objeto Rectangle:

Lienzo.DrawRectangle(Lápiz, New Rectangle(100, 100, 200, 100))

Similar a DrawLines, disponemos del método DrawRectangles, que espera un array de rectángulos y los dibuja todos a la vez.

**Elipse**

El método DrawEllipse toma los mismos argumentos que el método DrawRectangle, pero en vez de dibujar un rectángulo, dibuja una elipse circunscrita en él. Para dibujar un círculo, por tanto, hay que definir un cuadrado:

'Dibuja una elipse circunscrita en el rectángulo anterior

Lienzo.DrawEllipse(Lápiz, New Rectangle(100, 100, 200, 100))

'Dibuja un círculo

Lienzo.DrawEllipse(Lápiz, New Rectangle(200, 200, 200, 200))

El método natural de trazado de un círculo no es, sin embargo, su circunscripción en un rectángulo, sino el trazado de una línea equidistante en todos sus puntos de un punto central. Escribiremos, por tanto, una función que devuelva un rectángulo a partir del punto central y sus ejes mayor y menor, que además nos servirá para definir no sólo el círculo, sino cualquier elipse a partir del punto central y sus dos ejes:

Private Function DameRec(ByVal PuntoCentral As Point, \_

ByVal EjeMayor As Integer, \_

ByVal EjeMenor As Integer) \_

As Rectangle

Rem Devuelve un rectángulo a partir de su punto central y sus dos ejes

Dim Punto As Point = New Point(PuntoCentral.X - EjeMayor, PuntoCentral.Y - EjeMenor)

Dim Tamaño As Size = New Size(EjeMayor \* 2, EjeMenor \* 2)

Return New Rectangle(Punto, Tamaño)

End Sub

Private Sub DibujaElipse(ByVal Lienzo As Graphics, \_

ByVal Lápiz As Pen, \_

ByVal Punto As Point, \_

ByVal EjeMayor As Integer, \_

ByVal EjeMenor As Integer)

Lienzo.DrawEllipse(Lápiz, DameRec(Punto, EjeMayor, EjeMenor))

End Sub

Private Sub DibujaCírculo(ByVal Lienzo As Graphics, \_

ByVal Lápiz As Pen, \_

ByVal Punto As Point, \_

ByVal Radio As Integer)

DibujaElipse(Lienzo, Lápiz, Punto, Radio, Radio)

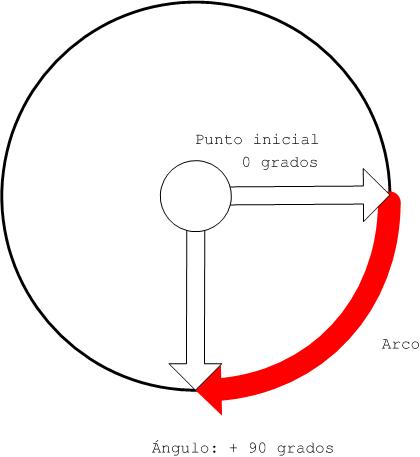
End Sub

**Arco**

Dibujar un arco es igual que dibujar un círculo, añadiéndole dos argumentos más: el punto inicial del arco y el ángulo que forma girando en el sentido de las agujas del reloj, ambos medidos en grados. El método se llama DrawArc:

gr.DrawArc(New Pen(Color.Red, 10), New Rectangle(250, 250, 250, 250), 0, 90)

es decir, estamos dibujando un arco cuyo punto inicial es el punto 00 y se extiende 900 en el sentido de las agujas del reloj, como muestra el siguiente esquema:



Polígono

El método DrawPolygon recibe, además del lapicero, un array de puntos, y los une con rectas. Hasta aquí es igual al método DrawLines. La diferencia está en que DrawPolygon une el primer punto con el último del array formando así un polígono cerrado. El siguiente ejemplo dibuja un rombo:

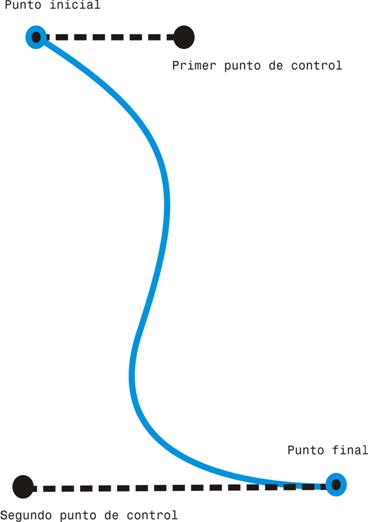
Lienzo.DrawPolygon(Lápiz, New Point() {New Point(200, 50), New Point(300, 100), New Point(200, 150), New Point(100, 100)})

**Curva Bézier**

Una curva Bézier está compuesta por sus puntos inicial y final y dos puntos de control. La función de los puntos de control no es que la curva pase por ellos, sino influenciar su dirección. Lo veremos con un ejemplo. La línea de color azul es la curva Bézier. Cada punto de control está unido por una línea imaginaria con su punto: el primero, con el inicial de la curva, y el segundo con el final. Pues bien, la curva inicia su recorrido en la dirección de la primera línea imaginaria, pero, a medida que avanza, rectifica su dirección para adecuarse a la que le marca la segunda línea imaginaria. Así es como se forma la curva. El dibujo del ejemplo está trazado a mano, por ello quizá no sea exacto. Lo mejor para entender las curvas Bézier es experimentar con ellas.

El método que dibuja curvas Bézier se llama DrawBezier. Toma como parámetros, además del lapicero, los cuatro puntos: el punto inicial de la curva, el primer punto de control, el segundo punto de control y el final de la curva:

Lienzo.DrawBezier(New Pen(Color.Red, 10), New Point(10, 30), New Point(100, 20), New Point(140, 190), New Point(200, 200))



**Curvas cardinales**

Se llaman así las curvas definidas por un valor numérico llamado “tensión” que mide la flexibilidad de la curva. Imaginemos dos puntos unidos por una recta. Si esta recta se convierte en curva, la longitud del arco y, por tanto, la “profundidad” de la curva aumentan según aumente la “flexibilidad” de la línea. Una línea recta equivale a una tensión máxima representada por el valor 0, y una línea curva disminuye su tensión y aumenta su flexibilidad hasta un valor de 2. Lo mejor para entenderlo es experimentar. El método que dibuja líneas cardinales se llama DrawCurve. El dibujo muestra el resultado del siguiente código:

Dim Puntos() As Point = {New Point(100, 100), New Point(400, 400), New Point(450, 130), New Point(650, 200)}

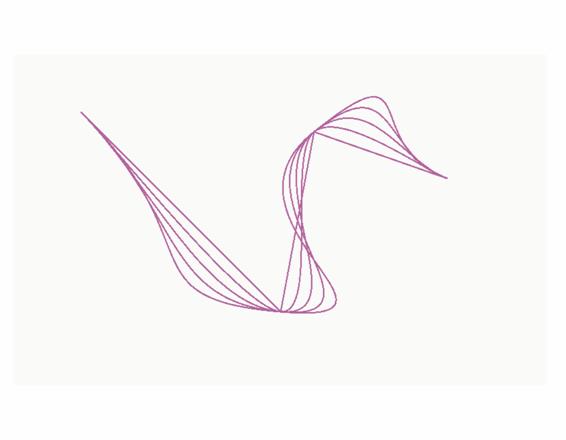
Dim tensión As Single 'tensión

For tensión = 0 To 2 Step 0.5

Lienzo.DrawCurve(New Pen(Color.Magenta, 2), Puntos, tensión)

Next

Según el valor de la tensión se acerca 2, la línea se va curvando más.

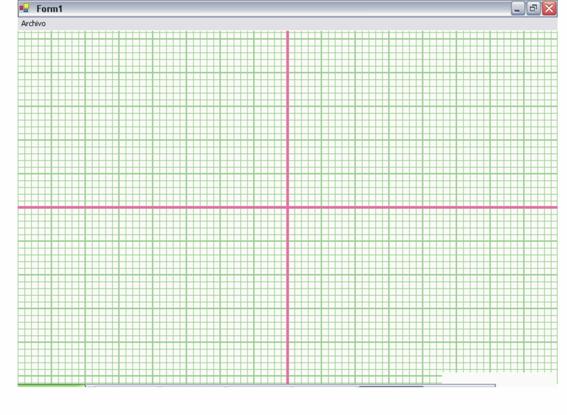


**Trayectos**

Sería muy interesante construir una clase que generara un deteminado dibujo para poder plasmarlo cuantas veces quisiéramos y donde quisiéramos. VSNET nos ofrece una. Se trata de GraphicsPath. GraphicsPath tiene estructura de colección, de tal modo que podemos añadir y/o eliminar a discreción las formas componentes del dibujo. El dibujo resultante queda así encapsulado dentro de un objeto de tipo GraphicsPath.

Además, GraphicsPath nos regala una capacidad añadida: el punto inicial de cada forma se conecta con el final de la forma anterior, a menos que lo impidamos invocando al método StartFigure(). Por eso se llama Path (camino, ruta), porque, salvo prohibición vía código, todas las formas se conectan formando una línea que las atraviesa a todas, formando su trayectoria, de ahí mi traducción de “trayecto”.

El siguiente ejemplo muestra la potencia de este objeto al reutilizarse para dibujar un eje cartesiano de coordenadas sobre una superficie milimetrada. He aquí nuestro objetivo:



Y aquí el código:

Private Sub DibujaGrid()

Dim i As Short

Dim Intervalo As Integer

Dim Lienzo As Graphics = Me.CreateGraphics

'Definimos tres objetos GraphicsPath

'uno encapsulará los dos ejes

'otro encapsulará la cuadrícula gruesa y el tercero la fina

Dim Ejes As New GraphicsPath()

With Ejes

'Eje horizontal

'La diferencia entre Me.Height y Me.ClientSize.Height

'es que Me.Height mide la distancia entre los dos bordes,

'mientras que Me.ClientSize.Height toma en cuenta sólo el área cliente.

'El área cliente es la superficie del formulario en la que

'se pueden depositar controles. Por tanto, están excluidos

'del área cliente la barra de título y los bordes

.AddLine(New Point(Me.ClientSize.Width / 2, 0), (New Point(Me.ClientSize.Width / 2, Me.ClientSize.Height)))

.StartFigure() 'para que no una los dos ejes con otra línea

.AddLine(New Point(0, Me.ClientSize.Height / 2), New Point(Me.ClientSize.Width, Me.ClientSize.Height / 2))

End With

'el método DrawPath toma como argumentos

'un lapicero y un trayecto

'Aquí el trayecto lo devuelve la función Grid

'escrita a continuación

'Primero dibujamos la cuadrícula fina

Lienzo.DrawPath(New Pen(Color.LightGreen, 1), Grid(Me, 10))

Lienzo.DrawPath(New Pen(Color.LightGreen, 2), Grid(Me, 50)) 'luego la gorda

Lienzo.DrawPath(New Pen(Color.HotPink, 4), Ejes) 'y al final los ejes

'El orden es importante porque determina qué líneas pasan por encima de cuáles

Lienzo.Dispose() 'Al final cerramos el objeto Graphics

End Sub

Private Function Grid(ByVal Formulario As Form, ByVal Intervalo As Integer) As GraphicsPath

Dim i As Short

'Necesitamos saber el resto obtenido de dividir la anchura o la altura

'por la distancia entre cada línea recogida en el parámetro Intervalo

'Para ello el operador mod nos viene que ni pintado.

'Dividimos el resultado por dos para repartir el margen

'a los dos extremos de la cuadrícula

Dim MargenXInicial As Integer = (Formulario.ClientSize.Width Mod Intervalo) / 2

Dim MargenYInicial As Integer = (Formulario.ClientSize.Height Mod Intervalo) / 2

Dim Trayecto As New GraphicsPath()

With Trayecto

'Comenzando por el margen inicial, vamos dibujando líneas

'guardando entre ellas la distancia que nos indica la variable Intervalo

'Líneas verticales

For i = MargenXInicial To Me.Width - MargenXInicial Step Intervalo

.AddLine(New Point(i, 0), New Point(i, Me.ClientSize.Height))

.StartFigure()

Next

'Líneas horizontales

For i = MargenYInicial To Me.Height - MargenYInicial Step Intervalo

.AddLine(New Point(0, i), New Point(Me.ClientSize.Width, i))

.StartFigure()

Next

End With

Return Trayecto

End Function

**Transformaciones**

Transformar una forma, un GraphicsPath, una región o todo el objeto Graphics consiste en modficar su posición o su tamaño. En realidad la transformación no se aplica al dibujo, sino a su eje de coordenadas. Técnicamente, una transformación es una personalización aplicada al eje de coordenadas de mundo. Básicamente las tranformaciones son tres: rotacón, traslación y escalado.

**Rotación**

Consiste en el giro del eje de coordenadas sobre sí mismo en el sentido de las agujas del reloj. El eje del giro es el origen de coordenadas (0, 0).

En el ejemplo siguiente dibujamos un rectángulo y después giramos el eje de coordenadas 45º. El método que rota el eje de coordenadas se llama RotateTransform.

Private Sub Rotación()

Dim i As Short

Dim Lienzo As Graphics = Me.CreateGraphics

Dim TRAYECTO1 As New GraphicsPath()

Dim Esquina As New Point(250, 50)

Dim Tamaño As New Size(250, 100)

'Esta línea une el origen de coordenadas (0,0)

'con la forma. Se comprueba en el resultado

'que las dos líneas (la original y la transformada)

'forman un ángulo de 45º

TRAYECTO1.AddLine(New Point(0, 0), Esquina)

Dim Lápiz1 As New Pen(Color.LightGreen, 8)

Lápiz1.EndCap = LineCap.ArrowAnchor 'le añadimos una flecha

'Encapsulamos la línea en un GraphicsPath

'y el rectángulo en una región

'en vez de encapsular ambos en el mismo objeto

'para poder dibujarlos con colores diferentes

Dim Trayecto2 As New GraphicsPath()

Trayecto2.AddRectangle(New Rectangle(Esquina, Tamaño))

Dim Región1 As New Region(Trayecto2)

For i = 0 To 1

Lienzo.DrawPath(Lápiz1, Trayecto1) 'Dibujamos la línea

'Combinamos el GraphcsPath con la región

'para que la transformación se aplique

'a los dos objetos en conjunto

Región1.Union(Trayecto1)

'Dibujamos la región completa

Lienzo.FillRegion(Brushes.Blue, Región1)

'y aplicamos la transformación:

'rotación del eje de coordenadas

'45º en el sentido de las agujas del reloj

Lienzo.RotateTransform(45)

‘Después de personalizar el eje de coordenadas

‘hay que dibujar la forma, path o región

‘para que la transformación tenga efecto.

‘este bucle dibuja una región

‘aplicándole transformaciones sucesivas

Next

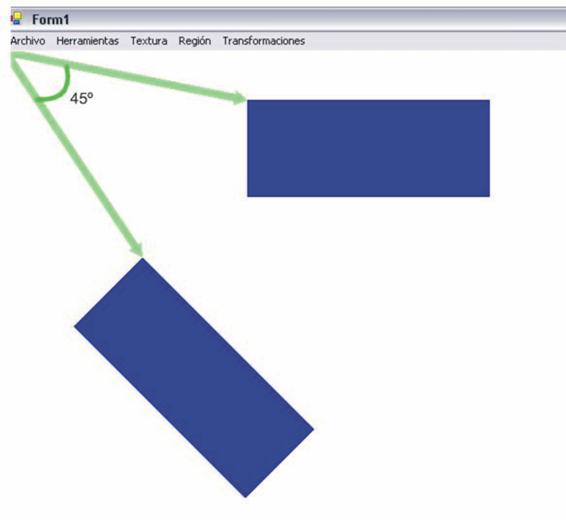
Trayecto1.Dispose()

Trayecto2.Dispose()

Región1.Dispose()

Lienzo.Dispose()

End Sub



**Traslación**

Esta transformacón traslada (de ahí su nombre) el origen de coordenadas a otro lugar, convirtiéndolo en el nuevo origen de coordenadas (0, 0). El método se llama TranslateTransform y toma como parámetro el nuevo punto que servirá de origen. Técnicamente hablando, TranslateTransform mueve el origen de coordenadas de mundo a un deteminado punto definido por el eje de coordenadas de página.

En el capítulo dedicado al GraphicsPath aprendimos a dibujar un eje cartesiano sobre una página milimetrada. Ahora que sabemos mover el origen de coordenadas (0,0), vamos a trazar la gráfica de la funcón seno sobre un eje de coordenadas cuyo origen es el centro del formulario:

Private Sub Traslación()

DibujaGrid() 'Llamamos al método expuesto más arriba

Dim Lienzo As Graphics = Me.CreateGraphics

Dim Trayecto1 As New GraphicsPath()

Dim m As Integer = 36 'Aleatorio

'el eje de abcisas (x) tendrá

'36 puntos negatvos y 36 positivos

'En este array almacenaremos

'los puntos que forman la gráfica de la función

Dim Puntos(m \* 2) As Point

Dim i As Short

For i = -m To m

'Hemos alterado un poco las coordenadas X e Y de los puntos

'para que la gráfica ocupe toda la pantalla y se aprecie mejor,

‘por tanto las unidades de medida de la rejilla y de la gráfica

‘no coinciden, pero eso ahora no tiene importancia.

'El valor Y lo multiplicamos por -1

'porque, si bien hemos movido el origen de coordenadas (0,0),

'no hemos alterado la disposición de los valores positivos y negativos.

'Queremos que los valores negativos del eje de ordenadas (y)

'aumenten hacia abajo, no hacia arriba, cual es el caso por defecto.

'Por eso tenemos que multiplicar el componente Y por -1

'en la función seno no se aprecia la diferencia,

'pero en otras la diferencia es obvia.

Puntos (i + m) = New Point(m / 3 \* i, -m \* 6 \* Math.Sin(i))

Next

'Construimos la línea curva

'y se la entregamos al GraphicsPath con AddBeziers

Trayecto1.AddBeziers(Puntos)

'Aquí movemos el eje de coordenadas

'al punto central del área cliente del formulario

Lienzo.TranslateTransform(Me.ClientSize.Width / 2, Me.ClientSize.Height / 2)

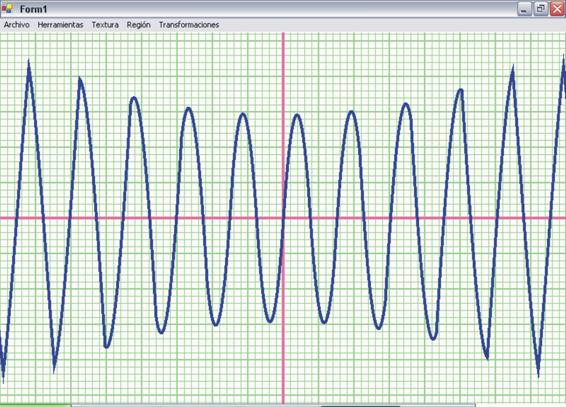
'Y dibujamos la gráfica

Lienzo.DrawPath(New Pen(Color.Blue, 4), Trayecto1)

Trayecto1.Dispose()

Lienzo.Dispose()

End Sub



**Escalado**

Escalar consiste en aumentar o disminuir la distancia entre valores consecutivos de la unidad de medida de los ejes de coordenadas. Técnicamente, el escalado es una operación realizada sobre la unidad de medida definida para las coordenadas de página y asumida por las coordenadas de mundo que consiste en la multiplicación de esta unidad de medida por un número, obteniendo otra unidad de medida totalemente personalizada. Además, pueden definirse unidades de medida diferentes para el eje de abcisas (X) y para el de ordenadas (Y). El efecto resultante son formas proporcionalmente más grandes o más pequeñas.

El método se llama ScaleTransform, y toma como parámetros los factores por los que se multiplican las unidades de medida vigentes para los ejes X e Y de las coordenadas de mundo. Visualmente es como si se multiplicaran por esos factores la base y la altura, respectivamente, del rectángulo que circunscribe el dibujo.

El escalado, además, no afecta sólo al tamaño de las formas, sino también al grosor del lapicero.

En definitiva, se puede comprobar que, puesto que la transformación se aplica a los ejes de coordenadas y no a las formas, aunque éstas varíen de tamaño, los valores que las miden no varían.

El siguiente ejemplo muestra un rectángulo escalado dos veces. Si observa el lector los valores que devuleven las propiedades Width y Height del rectángulo, comprobará que no varían:

Private Sub Escalado)

Dim Lienzo As Graphics = Me.CreateGraphics

Dim Trayecto1 As New GraphicsPath()

Dim Región1 As New Rectangle(10, 10, 100, 50)

Trayecto1.AddRectangle(Región1)

Lienzo.DrawPath(Pens.Blue, Trayecto1) 'Dibujamos un rectángulo sin transformar

MsgBox(Región1.Width) ’200

Lienzo.ScaleTransform(2.0F, 2.0F)

‘Una unidad en los dos ejes ha aumentado el doble

'El efecto es como si multiplicáramos

‘base y anchura del rectángulo por 2

Lienzo.DrawPath(New Pen(Color.Blue, 2), Trayecto1)

MsgBox(Región1.Width) ’200

Lienzo.ScaleTransform(3.0F, 4.0F)

‘Una unidad en el eje X ha aumentado el triple

‘Una unidad en el eje Y ha aumentado el cuádruple

'Efecto: multiplicar base por 3 y altura por 4.

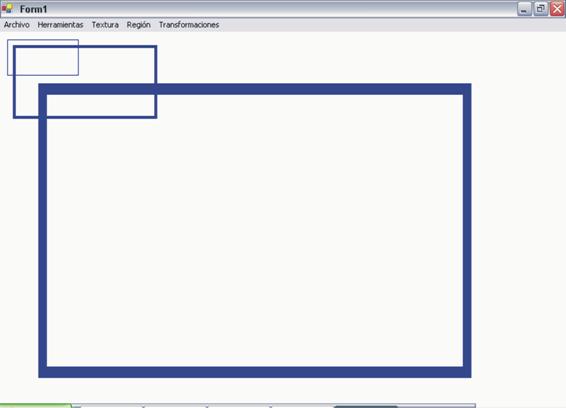
Lienzo.DrawPath(New Pen(Color.Blue, 2), Trayecto1)

MsgBox(Región1.Width) ’200

‘Se ve claramente en el dibujo

‘que el escalado también afecta al grosor del lapicero

End Sub



**Ámbito de las transformaciones**

Los tres ejemplos expuestos muestran transformaciones aplicadas a todo el área de dibujo, es decir, al objeto Graphics. Son las llamadas transformaciones globales o de mundo. Como dijimos al principio, las transformaciones también se pueden aplicar a regiones, GraphicsPath, o incluso brochas: son las transformaciones locales, que afectan sólo a un determinado objeto y no al resto. Veamos un ejemplo en el que dibujamos un pedazo de tarta y la tarta sin el pedazo, y aplicamos transformaciones diferentes a cada figura. Y aprovechamos el ejemplo para introducir el objeto Matrix:

Private Sub TexturaPredeterminada\_Click()

Dim Lienzo As Graphics = Me.CreateGraphics

Dim Brocha As TextureBrush = New TextureBrush(PictureBox1.Image)

Dim Trayecto1 As New GraphicsPath(),Trayecto2 As New GraphicsPath()

Dim Radio As Integer = 200

‘Definimos el punto central del formulario

Dim Centro As Point = \_

New Point(Me.ClientSize.Width / 2, Me.ClientSize.Height / 2)

'Trasladamos el origen de coordenadas al centro del lienzo

'Esta transformación es global porque

'afecta a todo lo que dibujemos a partir de aquí

Lienzo.TranslateTransform(Centro.X, Centro.Y)

'Añadimos la tarta sin el pedazo

Trayecto1.AddPie(DameCuadrado(New Point(0, 0), Radio), 90 \* 0, 90 \* 3)

'y le aplicamos una transformación local

‘(Hemos importado System.Math)

Trayecto1.Transform(New Matrix(Cos(45), Sin(45), -Sin(45), Cos(45), 0, 0))

'Añadimos el pedazo

Trayecto2.AddPie(DameCuadrado(New Point(0 + 20, 0 - 20), Radio), 90 \* 3, 90)

'y le aplicamos otra transformación local

Trayecto2.Transform(New Matrix(Cos(45), Sin(45), -Sin(45), Cos(45), 100, 25))

'Explicamos el objeto Matrix en el siguiente capítulo

Lienzo.FillPath(Brocha, Trayecto1) 'Aquí dibujamos

Lienzo.FillPath(Brocha, Trayecto2) 'cada GraphicsPath

Brocha.Dispose()

Lienzo.Dispose()

End Sub



**Movimiento**

Una de las aplicaciones más interesantes de GDI+ es la simulación de movimiento.

Todo se reduce a dibujar una figura, cambiarla de sitio, dibujar la figura en su

nuevo lugar y borrar la anterior. Para terminar este tutorial ofrecemos al lector un

sencillo osciloscopio que representa la función seno en movimiento. He aquí el

código:

Private Sub Movimiento\_Click()

Me.BackColor = Color.Black

Lienzo = Me.CreateGraphics

Trayecto = New GraphicsPath()

'Movemos el eje de coordenadas al centro del formulario

Lienzo.TranslateTransform(Me.ClientSize.Width / 2, \_

Me.ClientSize.Height / 2)

Dim m As Integer = 36 'Aleatorio

ReDim tr(m \* 2) 'Metemos todo el recorrido en una matriz de

puntos

Dim i As Short : For i = -m To m

tr(i + m) = New Point(m / 3 \* i, -m \* 6 \* Math.Sin(i))

Next

'Iniciamos el temporizador

With Timer1

.Interval = 100

.Enabled = True

.Start()

End With

Application.DoEvents()

End Sub

Private Sub Timer1\_Tick(ByVal sender As Object, ByVal e As

System.EventArgs) Handles Timer1.Tick

If index > tr.GetUpperBound(0) - 4 Then index = 0

'Borramos los dos últimos puntos

Lienzo.DrawCurve(New Pen(Me.BackColor, 1600), tp)

'Metemos los cuatro últimos puntos en el array

tt(0) = tr(index)

tt(1) = tr(index + 1)

tt(2) = tr(index + 2)

tt(3) = tr(index + 3)

'Reservamos dos para borrarlos

tp(0) = tt(0)

tp(1) = tt(1)

index += 1

Me.Text = String.Format("X = {0}; Y = {1}", tt(3).X, tt(3).Y)

'Dibujamos la curva

Trayecto.AddCurve(tt)

Lienzo.DrawPath(New Pen(Color.Yellow, 4), Trayecto)

Trayecto.Reset()

End Sub

Private Sub Stop\_Click()

Timer1.Stop()

Timer1.Enabled = False

End Sub

Private Lienzo As Graphics

Private Trayecto As GraphicsPath

Private index As Integer

Dim tr() As Point

Dim tt(3) As Point

Dim tp(1) As Point